**模拟考试**

|  |  |  |  |
| --- | --- | --- | --- |
| 题目名称 | 数字三角形 | 跳跃滴球 | Ex |
| 文件名称 | number | jump | expression |
| 输入文件名称 | number.in | jump.in | expression.in |
| 输出文件名称 | number.out | jump.out | expression.out |
| 时间限制 | 1秒 | 1秒 | 1秒 |
| 内存限制 | 256M | 256M | 256M |
| 题目分值 | 100分 | 100分 | 100分 |

数字三角形

**题目描述：**

让人怀念的数字三角形。

给出一个如下的数字三角形：

1

1 2 1

1 2 3 2 1

1 2 3 4 3 2 1

……

你从三角形的顶部出发，假如右边有数，则移动到右边相邻的数上，否则，移动到下一行第一个数上，沿途取走所有经过的数，现在多次询问，若要使取走的数的和大于等于N，则至少需要经过多少个数？

**输入：**

第一行输入T，代表询问的次数，之后T行输入每次询问的N。

**输出：**

输出T行，每行表示询问的答案

**样例输入：**

5  
6  
9  
11  
21  
35

**样例输出：**

5

7

7

13

19

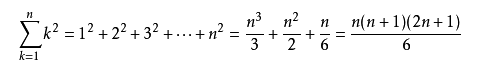
**数据规模：**

20%:T=1,N<=1000

40%:T<=10,N<=10000

100:T<=100000,N<=10^18

观察可得，数字三角形每行之和即为n^2，推出平方项的求和公式后二分，先确定所在的行，再确定该行的哪个数，在前一半还是后一半



#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<ctime>

#define ll long long

using namespace std;

int T;

ll n,ans;

ll Calc(ll x)

{

return (2ll\*x+1)\*(x+1ll)\*x/6ll;

}

ll calc(ll x)

{

return (x+1ll)\*x/2ll;

}

bool Check(ll x)

{

return Calc(x)<=n;

}

ll F(ll x)

{

if(!n)return 0;

if((x+1ll)\*x/2ll>=n)

{

ll l=1,r=x,mid;

while(l<r)

{

mid=(l+r)/2ll;

if((mid+1ll)\*mid/2ll>=n)r=mid;

else l=mid+1;

}

return l;

}

else

{

ll l=0,r=x-1,mid;

while(l<r)

{

mid=(l+r)/2ll;

if(1ll\*x\*x-calc(mid)>=n)l=mid+1;

else r=mid;

}

l--;

return 2\*x-l-1;

}

}

int main()

{

freopen("number10.in","r",stdin);

freopen("number10.out","w",stdout);

scanf("%d",&T);

for(int t=1;t<=T;t++)

{

scanf("%lld",&n);

ll l=1,r=1500000,mid;

while(l<r)

{

mid=(l+r)/2;

if(Check(mid))l=mid+1;

else r=mid;

}

n-=Calc(l-1);

ans=(l-1ll)\*(l-1ll)+F(l);

printf("%lld\n",ans);

}

return 0;

}

跳跃滴球

**题目描述：**

给你M个球，N个盒子，球与球，盒子与盒子之间互不相同，现在每个球都有预先指定的可以容纳它的两个盒子，你可以将这个球放进两个盒子中的任意一个。 现在，我们要求把所有的球放进盒子里，并且满足每个盒子里最多只有一个球，请求出有多少种满足要求的方案。

**输入：**

第一行输入两个数M,N，分别表示总共有M个球以及N个盒子

接下来输入M行，每行有两个整数x,y,（ 0<=x,y<N ）,表示该球所对应的两个盒子的编号，x可能等于y，这代表这个球只能被放在一个盒子里。

**输出：**

输出方案数对1 000 000 007 取模的结果,如果不存在一种合法方案，则输出0.

**样例输入1：**

4 5

0 1

1 2

3 4

4 3

**样例输出1：**

6

**样例输入2：**

3 2

0 0

1 1

0 1

**样例输出2：**

0

**数据规模：**

20%:N<=100,M<=15

40%:N<=1000,M<=1000

100:N<=100000,M<=100000

图上的归纳观察题目，首先，我们可以将每个球的两个地点连一条边，这样就形成了一张图。我们容易知道，最终题目的答案应该是每一部分连通图的放置方案数的乘积，那么现在问题转换成了如何计算一张连通图的放置方案数。

当一张n个点的图为一棵树时，其方案为n,因为我们只需要指定某一个点是空的，剩下的也就全部确定了。

当一张n个点的图存在且仅存在一个环时，若不为自环，则方案为2，即决策某条边上的决策，若为自环，则方案为1,这时方案本身已经确定了。

当一张n个点的图存在两个及以上的环时，边数就已经大于点数，显然没有合法方案。

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int mod=1000000007;

int n,m,cnt=0,sign=0;

int fa[100005]={0},sn[100005]={0},sm[100005]={0};

bool visit[100005]={0},h[100005]={0};

int Get(int x)

{

if(fa[x]==x)return x;

return fa[x]=Get(fa[x]);

}

int main()

{

scanf("%d%d",&n,&m);

for(int i=0;i<m;i++){ fa[i]=i; sn[i]=1; sm[i]=0; visit[i]=0; h[i]=0; }

int x,y,X,Y;

for(int i=1;i<=n;i++)

{

scanf("%d%d",&x,&y);

X=Get(x); Y=Get(y);

if(x==y){ sm[X]++; visit[X]=1; continue; }

if(X==Y){ sm[X]++; continue; }

fa[X]=Y;

sn[Y]+=sn[X];

sm[Y]+=sm[X]+1;

if(visit[X])visit[Y]=1;

}

int ans=1;

for(int i=0;i<m;i++)

{

x=Get(i);

if(h[x])continue;

h[x]=1;

if(sm[x]>sn[x]){ ans=0; continue; }

if(sm[x]==sn[x]-1){ ans=(1ll\*ans\*sn[x])%mod; continue; }

if(visit[x])continue;

ans=(ans\*2ll)%mod;

}

printf("%d\n",ans);

return 0;

}

**Ex**

**题目描述：**

给你一个原始的运算式，它由正整数和三种位运算符组成（&，|，^），比如说1&2|3,或者1|2|3，其中三种位运算符的优先级是一样的。现在呢，这个原始的运算式里的运算符有一定概率与它右边相邻的数字一起消失，比如说1&2|3，假设其中第2个位置的&有0.5的概率和第3个位置的2一起消失，第4个位置的|有0.4的概率和第5个位置的3一起消失，那么总共就有4种可能，第一种，只剩一个1 = 1，概率P1=0.5\*0.4=0.2，第二种，剩下1&2 =0,概率P2=(1-0.5)\*0.4=0.2,第三种，剩下1|3 =3,概率P3=0.5\*(1-0.4)=0.3,第四种，剩下1&2|3 = 3,概率P4=（1-0.5）\*（1-0.4）=0.3，故期望得到的运算式的结果为P1\*1+P2\*0+P3\*3+P4\*3=2。

现在，给你这样一个运算式，以及运算式中每个位置上的运算符与它右边相邻的整数一起消失的概率，请求出它期望得到的结果是多少。

**输入：**

第一行为一个正整数n ( 0 < n <= 233 )

第二行为这个式子里从左往右的n+1个整数Ai(其中Ai小于2^20)，  
第三行为该式子中从左往右的n个位运算符，

第四行为n个0到1之间的浮点数，分别表示每个位置上的运算符与它右边相邻的整数一起消失的概率

**输出：**

输出期望值结果，保留到小数点后4位。

**样例输入1：**

2

2 3 5

^ ^

0.1 0.2

**样例输出1：**

3.6600

**样例输入2：**

2

1 4 11

^ ^

0.5 0.5

**样例输出2：**

7.5000

**部分分**

对于40%的数据保证n<=15

套路，按位拆开之后DP即可：

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

int n,cnt=0;

int a[505]={0};

char s[505];

double p[505];

double Get(int x)

{

double b[300];

if(a[0]&(1<<x))b[0]=1;

else b[0]=0;

for(int i=1;i<=n;i++)

{

if(a[i]&(1<<x))

{

if(s[i]=='|')b[i]=p[i]\*b[i-1]+(1-p[i]);

if(s[i]=='&')b[i]=p[i]\*b[i-1]+(1-p[i])\*b[i-1];

if(s[i]=='^')b[i]=p[i]\*b[i-1]+(1-p[i])\*(1-b[i-1]);

}

else

{

if(s[i]=='|')b[i]=p[i]\*b[i-1]+(1-p[i])\*b[i-1];

if(s[i]=='&')b[i]=p[i]\*b[i-1];

if(s[i]=='^')b[i]=p[i]\*b[i-1]+(1-p[i])\*b[i-1];

}

}

return b[n];

}

int main()

{

cin>>n;

for(int i=0;i<=n;i++)cin>>a[i];

for(int i=1;i<=n;i++)cin>>s[i];

for(int i=1;i<=n;i++)cin>>p[i];

double ans=0;

for(int i=0;i<=20;i++)ans+=(1<<i)\*1.0\*Get(i);

cnt++;

printf("%.4lf\n",ans);

return 0;

}